スキーマ変換の容易なデータベースシステムのためのファイル構成(モデル表現とその構築に関する理論と実際の研究)

AUTHOR(S):

上林, 弥彦; 小島, 功; 矢崎, 朋夫; 矢島, 脩三

CITATION:

上林, 弥彦 ...[et al]. スキーマ変換の容易なデータベースシステムのためのファイル構成(モデル表現とその構築に関する理論と実際の研究). 数理解析研究所講究録 1983, 495: 326-345

ISSUE DATE:

1983-06

URL:

http://hdl.handle.net/2433/103584

数理解析研究所講究録
495巻 1983年 326-345

スキーマ変換の容易なデータベースシステムのためのファイル構成

上林 弥彦 (Yahiko KAMBAYASHI)
小嶋 功 (Isao KOJIMA)
矢崎 朋夫 (Tomoo YAZAKI)
矢島 脩三 (Shuzo YAJIMA)
(京都大学 工学部)

1. まえがき

近年，大規模な関係データベースシステムが市販されるようになり，マイクロプロセッサを用いた知能端末が低価格化してきたことから，これらを組合せて強力な利用者志向のシステムを容易に実現できるようになってきつつある。マイクロプロセッサによる知能端末を用いることで，高レベルの言語インタフェース（グラフィック言語，自然言語等）などの利用者向きの機能や，種々の対話的な機能を知能端末上で実現することが考えられる。特にデータベースの設計や解析には，システムと利用者とのひんぱんな対話が必要であるため，このような機能は重要である。

現在，本研究室では，次のようなデータベース準備機能を持つ関係データベースシステムを開発中である。

(1) データベーススキーマの設計と解析

(2) データベーススキーマの変換

このような機能は，一般の関係データベースシステムでは支援されておらず，従ってこれらの機能を従来のファイル構成で扱うと処理コストは非常に大きくなると考えられる。

本稿では，このような機能を効率よく支援し、マイクロコンピュータ上での実現に適したファイル編成について考察している。

2. データベース準備機能を有する関係データベースシステム

本システムは，従来机上で行なわれたデータベースの準備を支援しようとするもので，次のような手順でスキーマを設計することができる。（スキーマの逐次的設計）

(I) 与えられたサンプルデータより，初期スキーマを設計する。

(II) スキーマを評価し，その結果に従ってより良いものに変換する。

(III) 適切なスキーマを得るまで(II)を繰り返す。

本システムでは，拡張した関係モデルによるスキーマ設計を行ない，異なるモデル間のスキーマ変換を避けている。また，強力なスキーマ変換機能を持つため，一種の発展型データベースシステムとも考えられる。本システムでは、次

に示すようなデータ準備機能を支援している。

(a) 意味制約の検査

本システムでは、次に示すような意味制約が検査できる。制約が満たされない場合、例外となる関係の組が出力される。

(a-1) 関数従属性　　(a-2) 結合従属性

(a-3) 定義域従属性　　(a-4) 空値の検査

(b) 強力なデータベース操作

データベースの解析を行なうために、従来のシステムでは支援されていなかった以下のような操作を支援している。

(b-1) outer-join　　(b-2) 集合値の操作

(b-3) 強力な部分マッチ機能 (b-4) 種々の空値の扱い。

(c) 関係の正規化、非正規化の機能

実世界のデータから関係スキーマを容易に得るためには、次のような特徴を持つデータモデルを扱う必要がある。

(1) 実データの入力が容易に行なえるような柔軟なデータ構造を有すること。

(2) 構造性を有し、直観的な意味の把握が行ないやすいこと。

(3) 通常の関係モデルとの変換が連続的に行なえ、その操作が明確に定義されていること。

(4) 通常の関係モデルと意味制約の種類と表現能力に大きな差がないこと

本システムでは，属性値に集合値を許した非正規関係モデルを採用している。

(d) スキーマ変換の機能

(d-1) 属性の構成と分解の機能

(d-2) 関係名，属性名，属性値の変更機能

(d-3) スキーマ情報とデータ値の相互変換

(d-4) 基底関係の変更と意味制約の変更機能

以上示した機能によって，スキーマ変換やデータベースの解析を行なうことができる。

3. ファイルの構成

2. で示したような機能は，従来の関係データベースシステムでは支援されておらず，その操作は非常にコストがかかると考えられる。従来，データベースのスキーマに反映させる情報は安定なものと考えられたが，本システムでは，データ自身が動的に変化するので，このような変化に対応できる必要がある。一般によく用いられる索引による構成は，元となるデータが変化した場合対応できず，またその再構成も多くのコストがかかる。これに対し，本システムでは効率よいスキーマ操作を実現するために，データ値とスキーマ情報とを分離し，それぞれ定義域辞書と圧縮関係表に納めている。定義域辞書は，関係・属性にまたがって同じ値を1つ

にまとめ定義域単位に1つのファイルを作るもので，次のような特徴を有している。

(1) 定義域単位に同じ値が1つにまとめられ，データ量が減少するので，ディスクアクセスの回数が減らせる。

(2) 結合属性は同じ定義域辞書に含まれるので，意味のない結合などを検出できる。

(3) 文献のキーワードのように，同義でも異なる表現をとる語が入るのを避けることができる。

(4) 辞書の変更のみで，数字，文字以外のデータにも容易に拡張できる。

圧縮関係表は定義域辞書へのポインタで構成され，索引の代わりとなる圧縮値を記憶している。

(1) 各圧縮値及びポインタは固定長であり、データベース操作を単純かつ効率的に行なえる。

(2) 圧縮関係はデータ量が小さく，スキーマ変換などに伴なう再構成も比較的容易に行なえる。

定義域辞書を用いる一つの欠点は，ひんぱんな辞書の参照が処理効率の低下を招くことである。本システムでは多くの種類の圧縮値を用い，ほとんどのデータベース操作は定義域辞書を調べる前に圧縮値による擬似操作により行なわれ，辞書への参照は，最終段階での検査のみに用いられる。圧

縮値の利用は部分マッチ演算の効率化にも適している。

次に、ファイル構成の詳細について述べる。

(1) 定義域辞書　定義域辞書は、(図3.1)のような構成をとっている。各定義域値は可変長データが許され、区切記号"$"で区切られている。CP/Mの制限上、128バイト単位のレコードアクセスを行なうが、現在のところ一つの定義域値が複数のレコードにまたがることは禁止している。定義域値の検索は、圧縮関係からの直接アクセスを主としており、各定義域値の識別子として、次のようなIDを与える。(Domain ADDress 図3.2)

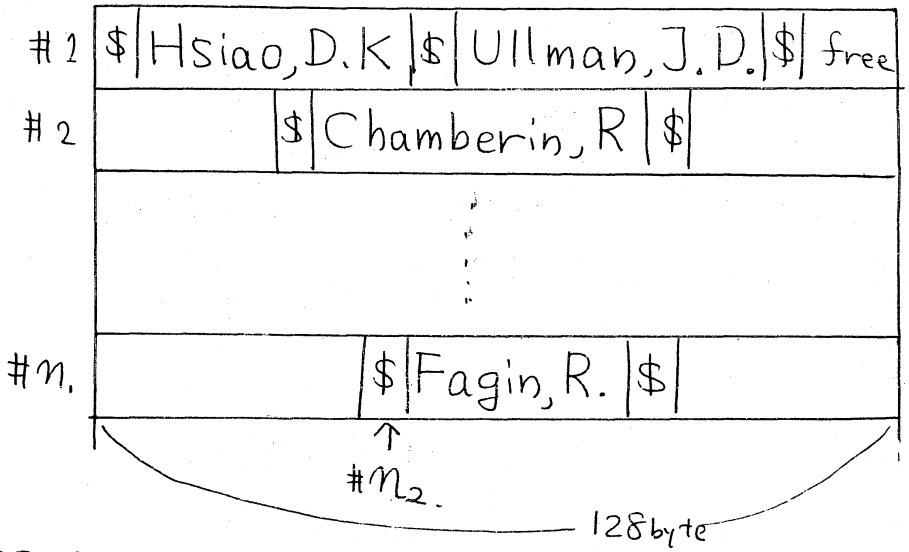
DADD :(1) 定義域値の含まれるレコード番号(2byte)
:(2) レコード中の定義域値の開始位置(6bit)

このような構成から、レコードの変更、削除は行ないにくい。また、ファイルはソートされていない。

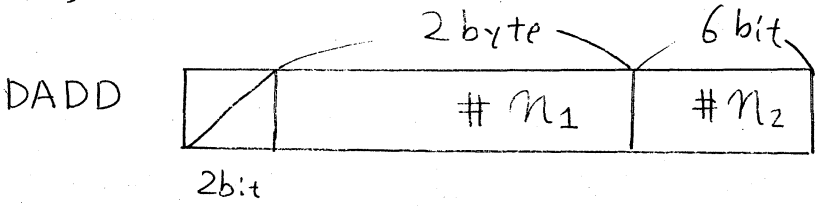
(2) 圧縮関係表　非正規関係を扱う圧縮関係は、(図3.3)のような構成をとっている。可変個の属性値を扱うため、各属性値は4byteの固定長で表わされている。1つの属性値は、次の3つから構成されている。

属性値 :(1) DADD (2byte + 6bit)
:(2) 属性値のハッシュ値(2byte)
:(3) 区切記号 (2bit)

(図3.1) 定義域辞書の構成



(図3.2) DADD



(図3.3) 圧縮関係ファイルの構成

RP	DADD a	Hash a	AP	DADD b	Hash b	VP	DADD c	Hash c	TP	DADD d	Hash d	AP	DADD e	Hash e

A	B
a	b c
d	e

RP : 関係の区切り(始まり)
AP : 属性の区切り(〃)
VP : 集合値の区切り(〃)
TP : 組の区切り(〃)

本システムで用いられているハッシュ値には、次のような種類がある。

(A) 文字列ハッシュ (図3.4)

(i) 定義域値の頭2文字をそのままハッシュ値とするもの。

(ii) 定義域値の各文字に対応して、2byte中の特定のビットを1にセットするもの。

(iii) 定義域値の頭文字と、(ii)の形式を1byteで実現したものとの連結

(B) 数値データ

(i) 整数型2byte：この場合、2byteの数値は圧縮する必要がなく、DADDを無視し、すべての計算は圧縮関係上で行なわれる。

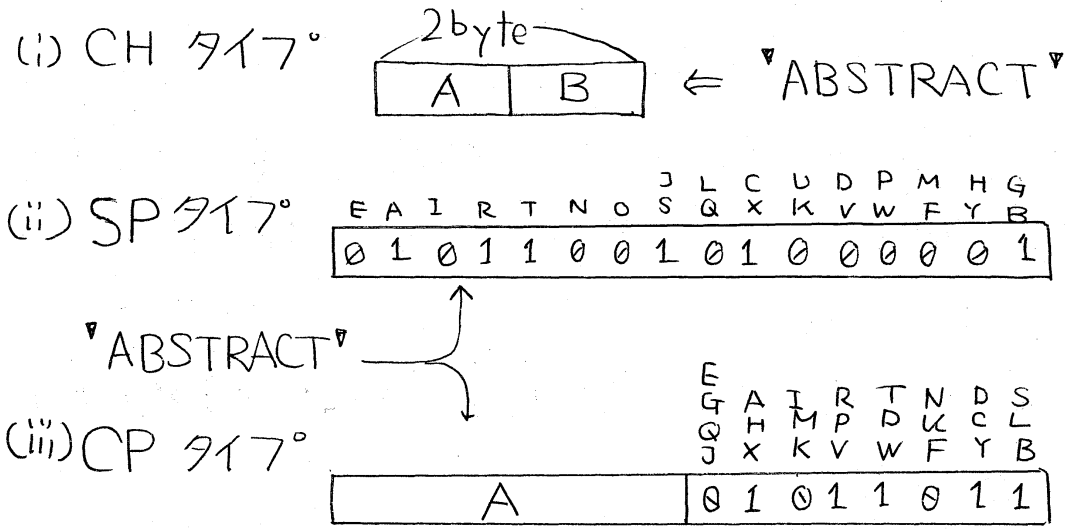
(ii) 長整数型 …… DADDを無視し、4byte + 6bit で表現する。この場合、特別な処理を必要とするが、現在のシステムでは言語上の制約から実現されていない。

これらのハッシュ法の決定は、関係の定義を行なう段階で、データ型として利用者が行なえる。

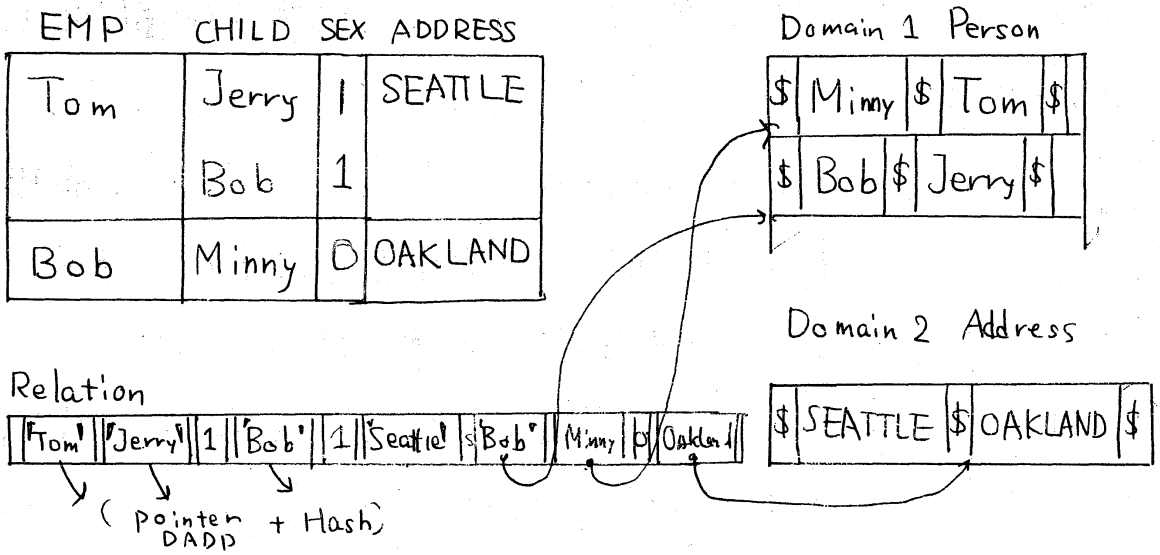
区切り記号は、関係/組/属性/値 によって図3.3の4種類がある。圧縮関係に対する操作は、順アクセスを主体とし、各区切り記号によって処理を制御している。

空値 本システムでは、outer-joinや不完全情報を扱う

(図3.4) ハッシュ値の構成



(図3.5) 関係表の構成



ために，種々の空値を扱うことが必要となる。 空値の種類と，その実現方法の例を次に示す。

(1) ?(0)非存在　(2) ?(t)未知　(3) ?(*)非存在か，未知かも不明，　(4) ?(i)個数がi個

[(1)～(4)はすべてDADDが0であり，定義域辞書はもたない。 空値の種類は，ハッシュ値により区別する。]

(5) ?(i－j)　個数がi～j個の間

(6) ?(i,……,j)　個数の可能性がi,…jのどれか

(7) ?(…)n　等しい空値がn個ある。

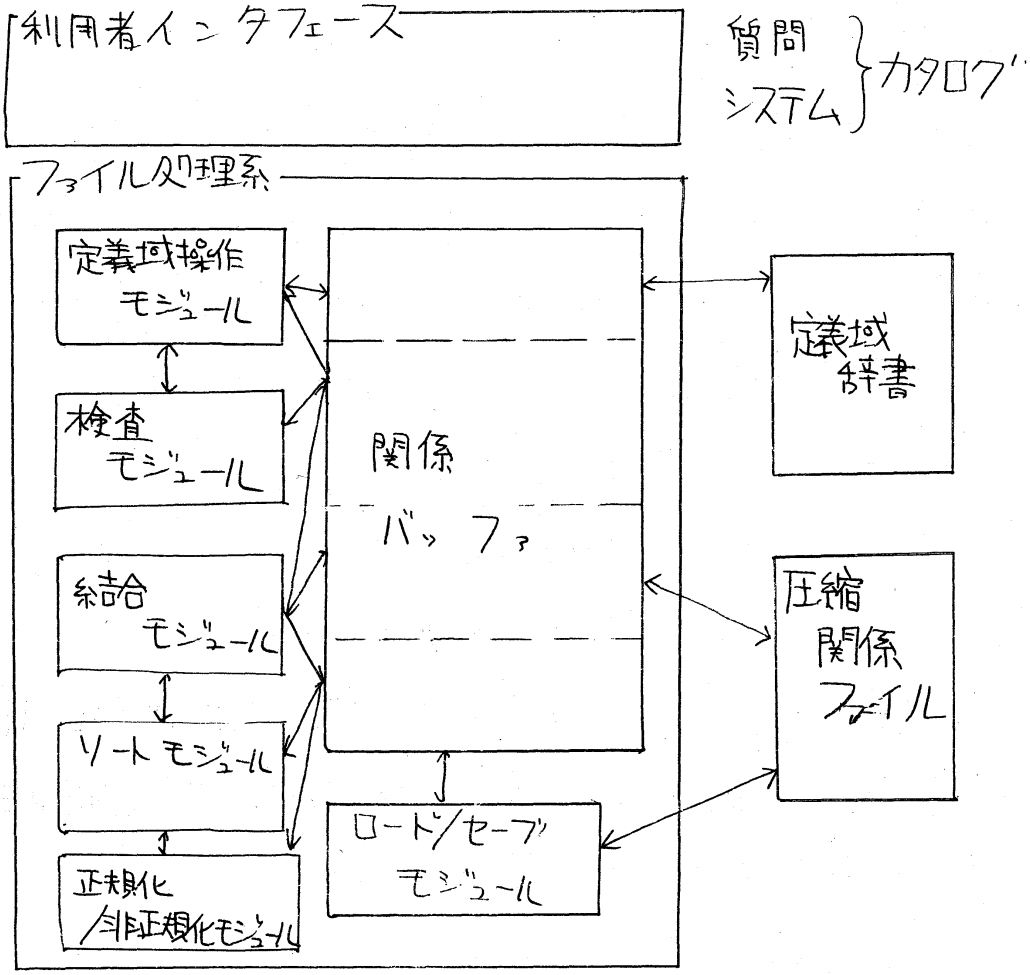
[この場合，空値に対して与えられる定義域辞書"null"を用い，辞書に空値の種類，等しい空値の個数を記憶する。]

(図3.5)に関係表の例を示す。

4. システムの概要

本システムはZ80ベースのマイクロコンピュータ上で言語Cを用いて開発されている。 システムの概要を(図4.1)に示す。 システムは，SQLを拡張したNF－SQLを利用者言語としている。 NF－SQLの特徴は(図4.2)のようなものである。 NF－SQLのコマンドに従ってファイル処理系の必要なモジュールが起動され，圧縮関係，定義域辞書等の操作を行なう。 本システムでは，メモリの一部の領域をバッファとして用い，3つまでの関係を格納することができ

(図4.1) システムの概要



(図4.2) NF-SQLの特徴文

(1) 分かりやすい言語 → SQLベース

(2) 質問の部分実行が可能 → 入れ子の禁止とJoinコマンドの独立，質問ブロックの導入

(3) QBEのような高レベルインタフェースが実現可能

(4) データ準備機能の実現 → 非正規関係の扱い
→ 強力なスキーマ変換操作の実現

る。　また、各操作もこれらのバッファを利用し、効率的な処理の実現を図っている。

5. システムにおける質問処理

本節では、システムにおける処理方式について説明する。処理例としては、(図5.1)に示すような関係質問を考える。

(1) 制約　二次記憶にある圧縮関係をメモリ上のバッファへ転送する。　これは圧縮関係の順アクセスによって実現されるが、この時に質問中の制約条件式をハッシュ値により検査し(擬似制約)、条件を満たす組のみをバッファへ転送する。同時に、結合属性をソート用のテーブルへ移し、ソートテーブルからバッファ上の組へのポインタを付ける。

(2) 結合　最も単純な場合を説明する。　結合する2つの関係に対して(1)で作られたテーブルをソートする。　ソート後に2つのテーブルを順に調べ、結合できる組の対を新たなバッファへ移す。　結合は、等結合の場合はDADDで、不等結合の場合はハッシュ値で行なう。(擬似結合)

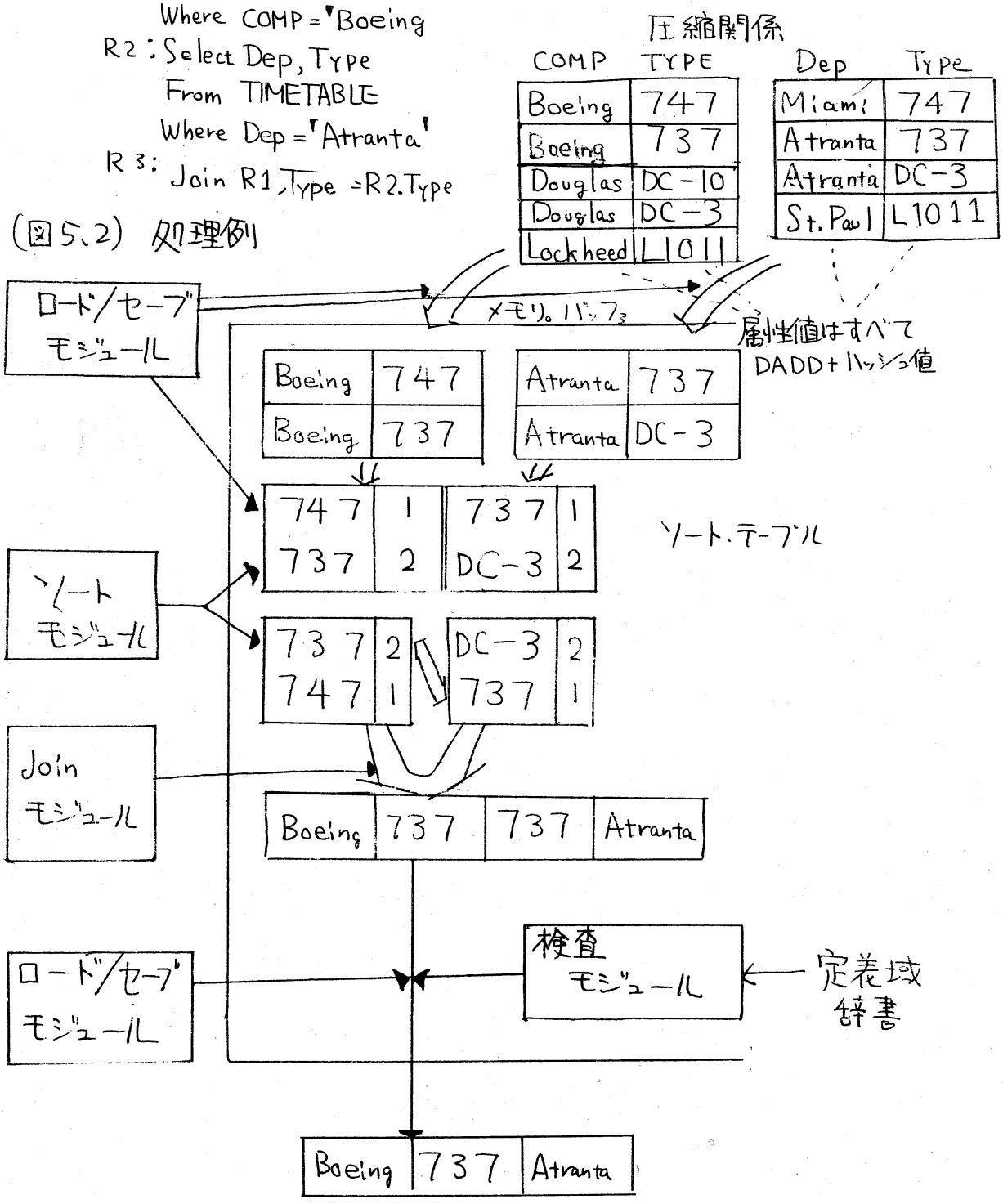
(3) 射影　結合結果のバッファから、必要な属性のみ抽出して、二次記憶へ格納する。　この段階で、定義域ファイルから実際の値を読み、擬似操作の検査を行ない正しい答のみ格納する。　同時に、組の重複も除く。

(1)～(3)の処理例を(図5.2)に示す。　スキーマ変換等の

(図5.1) 質問例

```
R1: Select COMP, TYPE
    From AIRCRAFT
    Where COMP = 'Boeing
R2: Select Dep, Type
    From TIMETABLE
    Where Dep = 'Atranta'
R3: Join R1.Type = R2.Type
```

(図5.2) 処理例



操作も同様に行なうが，いずれの場合もデータの操作は圧縮関係を用い，定義域辞書中のデータはできるだけ操作しないよう考慮している。また、圧縮関係は比較的小容量であるから，種々のスキーマの再構成も容易に実現できる。

6. ファイル操作の評価

6.1 質問処理方式の評価

ここでは，メモリバッファを利用した結合方式の比較を行なう。本システムでは，索引の代わりに圧縮値を用いるのと、メモリ容量が限られている点が従来の方法と異なっている。ここでは，次の6つの結合方式を考える。

(1) ソート・マージ(a) 5.の処理例で示した方法であり、制約条件を満たす組をすべてバッファに入れ，結合属性をソートテーブルに移す方法である。組の長さが可変であるから，ソートテーブルを用いる方が容易にソートできる。

(2) ソート・マージ(b) (1)の変形で，バッファ内はソート属性+DADDのみ残し、結合結果の組はあらためて関係表を直接アクセスして作成する。バッファをすべてテーブルに用いるので，より多くのデータが扱える。

(3) Nest・Loop(a) (1)と同じであるが，ソートテーブルを作らずに，組の組合せをすべてバッファ上で調べる方法である。

340

(図6.1) 処理方式の評価

C_S：二次記憶の順アクセスコスト　F_1, F_2：圧縮関係の大きさ

C_D：〃　直接　〃　　N_1, N_2：関係の組の数

S_S：制約によって残る組の比率　B：ブロックサイズ

S_J：結合　　A：属性数

$S_S = 1$ の場合の処理コストを示す。

(1) $C_S \times \frac{F_1 + F_2}{B}$　　(2) $C_S \times \frac{F_1 + F_2}{B} + C_D \times S_J (N_1 + N_2)$

(3) $C_S \times \frac{F_1 + F_2}{B}$　　(4) $C_S \times \frac{F_1}{B} + C_D \times \frac{F_2}{B}$

(5) $C_S \times \frac{F_1}{B} + C_D \times \frac{F_2}{B}$　　(6) $C_S \times \frac{F_1}{B} + C_D \times \frac{F_2}{B} + C_D \times S_J \times N_1$

また，扱える上限のレコード数については，バッファサイズがMの場合，

(1) $\frac{M}{5(A+1)}$　(2) $\frac{MA}{5}$　(3) $\frac{M}{5A}$　(4) $\frac{2M}{5A}$　(5) $\frac{2M}{5(A+1)}$　(6) $\frac{2MA}{5}$

(図6.2)

$C_S = \frac{1}{10} C_D = 1$

$F_1 = F_2$

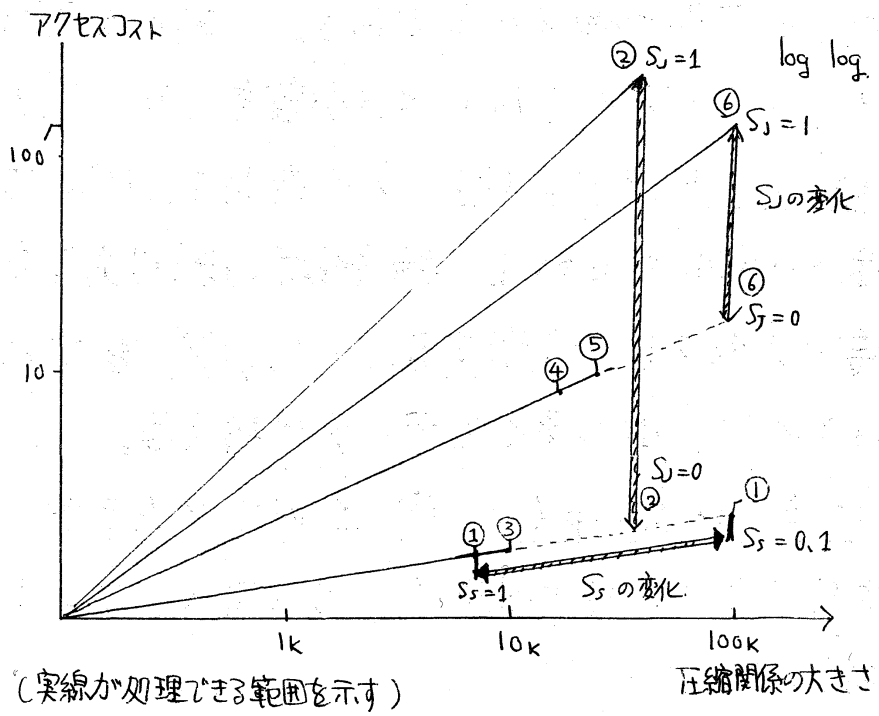
$N_1 = N_2$

$N_1 = \frac{F_1}{5A}$

$B = 256$

$A = 5$

$M = 10K$



(実線が処理できる範囲を示す)

(4) Nest. Loop(b)　(3)において，片側の関係のみバッファへロードし，相手側の関係を順に読むに従って結合を行なう。

(5) Nest・Loop(c)　(4)において，バッファにロードした関係にソートテーブルを設けるもの

(6) Nest・Loop(d)　(5)において，ソートテーブルのみバッファへロードするもの。　(2)と同じく結果はあらためて関係表へ直接アクセスする。

以上の方法を，二次記憶へのアクセス回数と扱えるレコード数の面から比較すると(図6.2)に示すような結果を得る。(2),(6)は結果のレコード数により処理コストが異なるが，$G_J = 1$において(2)の方がコスト大になる理由としては，(2)が直接アクセスを中心にしていることによる。現在のところ，(1)の方式を支援しているが，実際にシステム内で処理を行なう場合，これらの方式間のコスト差が大きいので，1つの方法のみ支援するのは問題がある。　また，これらの評価は$S_S = 1$の場合のみであり，これが変化しても扱えるレコード数は大幅に変わる。

そのため，これらの問題に対して，次のように動的に対応することが考えられる。

(1) まず第1の関係を制約しながらバッファへロードする。

バッファがいっぱいになれば，テーブルだけ残して結合属性とDADDのみ残し，(6)を実行する。結合属性もロードしきれない場合は，分割して処理をする。

(2) 第1の関係が入れば第2の関係をバッファへのロードする。全部入らなければ，(4),(5)を実行する。全部入れば(1),(3)を実行する。((1)と(3)，(4)と(5)の区別はソート属性による。)

このように実行時に最適化を行なうことでほとんどの場合に対応できる。質問解析の段階では，制約結果の小さくなる方を関係1に選べばよく，めんどうな統計情報の管理も必要がない。

索引を持つファイル編成と比較すると，次のようなことが考えられる。

(1) 索引を利用した演算に関しては，圧縮関係を用いるよりも有利であるが，本システムのように発展的に新たな関係が次々と作られる場合には，各結果に索引を構成するコストは大きい。また，圧縮関係を利用すると，定義域辞書のデータは動かないので，より一層不利になると考えられる。

(2) スキーマ変換などの処理に関しては，索引の再構成を必要とするので，特にB木のような索引を作る場合，手順も複雑で非常に多くの処理を必要とする。

6.2 記憶容量の比較

次に，本ファイルの占める記憶容量について，次の3つの場合を比較した（結果は 図6.3）

(1) 正規関係との比較

・この場合，圧縮関係も正規形として実現する。 定義域辞書では重複値は1つしか記憶されないので，異なる値の比率 $Cf = 1 - C/N$（Nはレコード数，Cは異なる値の数）を導入する。

(2) 非正規関係との比較

非正規関係は重複した値をGroup-byするので，記憶容量の減少が考えられる。 特に，圧縮関係のみのGroup-byより関係表のGroup-byの方が有利といえる。 図は一属性をGroup-byした場合を示している。

(3) 索引を設けた場合

関係表のみ索引を設けた場合を比較した。

7. あとがき。

本システムのように，データ自身が安定せず，データに加えられる操作も多様な場合は，特定の属性にのみ注目して索引を設ける方法は適さないと考えられ，本稿で示したようなファイル構成を利用したのは意義があると思われる。 問題点としては，マイクロコンピュータ上に実現するため，メモリ容量が限られており，処理効率が必らずしもよくない場合

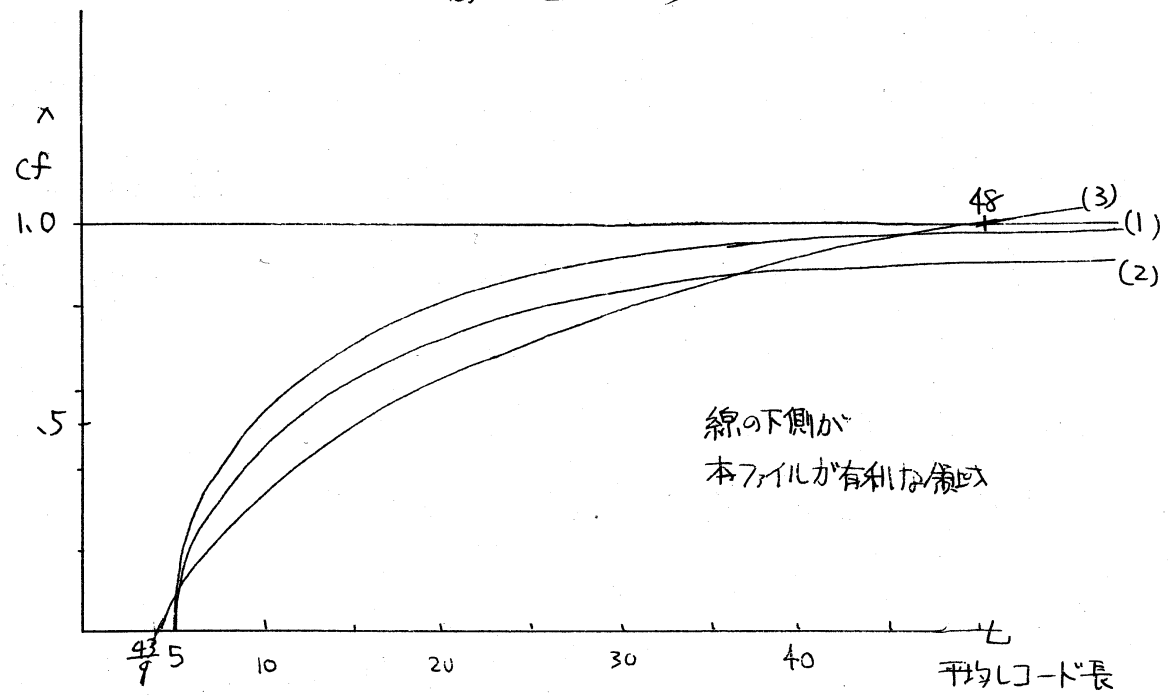
(図6.3)

(1) $Cf = 1 - \frac{C}{N}$　　平均レコード長 L，属性数 A とすると、

フラットテーブル　　圧縮関係 ⇒ N×A×(3+2)
⇒ N×A×L　　定義域辞書 ⇒ Cf×A×L×N　} 和

(2) 正規関係の減少量 ⇒ Cf×L×N　　(一属性)
圧縮関係の減少量 ⇒ Cf×5×N

(3) 索引による増加量 ⇒ (Cf×L+2)N (一属性)



があると考えられることと，定義域辞書の再構成の問題がある。

参考文献

[1] 上林，小島，矢崎，矢島「スキーマ設計及びデータ準備機能を有するマイクロコンピュータ関係データベースシステム」信学技報 EC83-5，1983年4月

[2] NCC "Panel on Evolutional Database Managent Systems," AFIPS NCC June 1979

[3] Blasgen,M.W. and Eswaran,K.P. "Storage and Access in Relational Data Bases",IBM Syst.J Vol.16 No.4 1977

[4] Yao,S.B. "Optimization of Query Evaluation Algorithms", ACM TODS Vol.4 No.2 1979

[5] Whang,K.Y. Wiederhold,G. and Sagalowicz,G. "Separability-An Approach to Physical Database Design", 7th VLDB Sept. 1981